SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 205-209, January 1989

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of this journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

BISECTION IS NOT OPTIMAL ON VECTOR PROCESSORS*

HORST D. SIMON†

Abstract. Recently there has been revived interest in the bisection method for computing eigenvalues of symmetric tridiagonal matrices, since this method lends itself easily to a parallel implementation. A natural extension of the bisection method is the multisection method. The relative advantages of these two methods have been discussed in several publications ([H. Bernstein and M. Goldstein, *SIAM J. Sci. Statist. Comput.*, 9(1988), pp. 601-602], [I. Ipsen and E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Tech. Report YALEU/DCS/RR-548, Yale Univ., Dept. of Computer Science, July 1987], [S. Lo, B. Philippe, and A. Sameh, *SIAM J. Sci. Statist. Comput.*, 8(1987), pp. s155-s165]). The purpose of this note is to contribute another argument in favor of using the multisection method, which did not arise explicitly in the past discussion. A simple analysis and some numerical examples show that the bisection method is in general not optimal in the class of multisection methods for the extraction of one eigenvalue on vector processors. Numerical results on a CRAY-2, a Convex C1-XP, and an Alliant FX/8 show that the optimal multisection section method can be several times faster than the bisection method.

Key words. symmetric tridiagonal eigenvalue problem, bisection method, multisection method, parallel bisection, vector processors

AMS(MOS) subject classification. 65F15

1. Introduction. Several researchers have recently investigated parallel algorithms for the symmetric tridiagonal eigenvalue problem [2], [5], [7], [8]. Dongarra and Sorensen [5] implemented the divide-and-conquer algorithm by Cuppen [3], and obtained considerable speedups over TQL2 from EISPACK [6],[12] on a CRAY X-MP/4 and an Alliant FX/8. Lo, Philippe, and Sameh [8] used a combination of bisection and inverse iteration and demonstrated an even better performance of their method than the divide-and-conquer algorithm on the Alliant FX-8, when computing all the eigenvalues and vectors. Ipsen and Jessup [7] investigated the same algorithms on a hypercube, and arrived at the conclusion that bisection and multisection are more efficient than Cuppen's divide-and-conquer method on distributed-memory parallel processors, with bisection faster than multisection.

The authors in [8] state that they prefer multisection over bisection in what they call the isolation phase. They give two reasons: (a) multisection creates more tasks than does bisection and thus achieves a better load balancing, and (b) there are several eigenvalues in one interval. In a recent timely communication Bernstein and Goldstein [1] argue against these two reasons for preferring multisection in the isolation phase. The purpose of this communication is to add another argument in favor of multisection to the current discussion. This argument has so far not been explicitly stated in any

* Received by the editors July 18, 1988; accepted for publication September 22, 1988.

† Numerical Aerodynamic Simulation (NAS) Systems Division, National Aeronautics and Space Administration Ames Research Center, Mail Stop 258-5, Moffett Field, California 94035. (The author is an employee of Boeing Computer Services.)

of the references: even for the computation of a single eigenvalue, bisection is not optimal on a vector processor.

In addition to introducing a new argument into the ongoing bisection versus multisection discussion, this observation has several other potential consequences. Two of the second-generation distributed-memory parallel computers, which have been introduced recently, provide vector processing [10],[11]. Thus the observation of this note also may change the outcome of Ipsen and Jessup's [7] numerical studies in favor of multisection, when carried out on a hypercube with vector processing at the nodes. Multisection is also an option considered in the design of LAPACK [4]. In [4] multisection is considered only as an option for a parallel environment. Based on the results here, multisection is also a valid option for eigenvalue extraction on a single vector processor.

2. Analysis of multisection on vector processors. Given a symmetric tridiagonal $n \times n$ matrix T, with

$$T \qquad \begin{array}{ccccc} \alpha_1 & \beta_2 & 0 & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & 0 \\ & & & & \\ 0 & & \beta_{n-1} & \alpha_{n-} & \beta_n \\ 0 & & 0 & \beta_n & \alpha_n \end{array}$$

The eigenvalues λ_i, $i = 1 \cdots n$ can be determined as the zeros of the so-called bottom-pivot function $\delta_n(\lambda)$, which is defined as follows:

$$\begin{array}{lll} \delta_1(\lambda) & \alpha_1 & -\lambda, \\ \delta_j(\lambda) & \alpha_j & \lambda - \dfrac{\beta_j^2}{\delta_{j-1}(\lambda)} \end{array}$$

The δ_j's are the elements of the diagonal matrix D in the LDL^T factorization of $T - \lambda I$. For a given point ς, the bottom pivot is commonly evaluated in the following loop (see [9]):

```
BISECTION ALGORITHM: $\delta \leftarrow \alpha_1 - \varsigma$
                     $\nu \leftarrow 0$
                     for $j = 2 \cdots n$ do
                         $\delta \leftarrow \alpha_j - \varsigma - \beta_j^2/\delta$
                         if $(\delta = 0)$ then $\delta \leftarrow \epsilon$
                         if $(\delta < 0)$ then $\nu \leftarrow \nu + 1$
                     end do
```

Here ϵ is a small multiple of the round-off unit of the machine used. Using Sylvester's inertia theorem, it follows that ν is equal to the number of eigenvalues less than ς (for details see [9]). Using ν the implementation of a bisection algorithm for computing an eigenvalue of T is straightforward.

Because of the nonlinear recurrence, the loop evaluating δ does not vectorize. However, if the value of ν is to be evaluated for p distinct trial points ς_k, $k = 1 \cdots p$, then the two loops can simply be interchanged. The additional cost are three arrays of the length p for the storage of the ς, δ, and ν values. This is trivial on today's vector machines. The benefit is that the evaluation of multiple δ's and ν's now vectorizes.

A simple analysis shows that on a scalar machine bisection is optimal, and no extra speedup is gained by using a multisection scheme. However, on a vector processor the simple loop interchange above allows us to evaluate ν for several points p at some higher speed. This may compensate for the additional arithmetic, which is performed by a multisection algorithm. That this is indeed the case will now be shown.

Suppose we want to locate an eigenvalue in the interval $[a_0, b_0]$, with length $l_0 = b_0 - a_0$, using a multisection algorithm with p points. At each step of a multisection

TABLE 1
Optimal number of multisection points.

$\frac{s}{r}$	p
0.0	1
1.3	2
2.6	3
4.1	4
5.8	5
7.7	6
9.7	7
11.8	8
14.1	9
16.4	10
18.9	11
21.5	12

algorithm, the length of the interval including the desired eigenvalue is reduced by a factor of $(p+1)^{-1}$. In order to reduce the initial interval to a length l_j, with $l_j/l_0 < \epsilon_0$, where ϵ_0 is some given tolerance, we thus have to carry out at least j multisection steps, with

$$j = \left\lceil \frac{-\log(\epsilon_0)}{\log(p+1)} \right\rceil$$

On the other hand, the evaluation of the recurrence for p points at each of the j steps can be carried out in vector mode in time proportional to

$$s + rp, \tag{1}$$

where s stands for the start-up cost of the vector loop and r the asymptotic rate. Hence the total cost C of the multisection algorithm is given by

$$C = (s + rp)\lceil \frac{-\log(\epsilon)}{\log(p+1)}$$

Ignoring the ceiling function for the moment, it easily checked that C is minimized for a value p_0 satisfying the equation

$$(p+1)\log(p+1) \quad p = \frac{s}{r} \tag{2}$$

Thus the optimal number of multisection points is a function of the ratio s/r, i.e., the ratio of start-up time and asymptotic rate for the loop described in the bisection algorithm above. Note that s/r is equal to the parameter $n_{1/2}$, which is used to denote the half-performance length in vector processing. Solving equation (2) for p one obtains Table 1, which lists the optimal number of multisection points as a function of s/r. For a start-up time $s = 0$, the known fact that bisection is optimal for scalar processors is recovered. For vector processors, we expect a value of s/r of about 10. This would indicate that a multisection algorithm with about seven or eight points is optimal on such a machine.

3. Numerical results. In order to validate the above analysis, a multisection algorithm has been implemented on several vector processors. In a first numerical experiment the inner loop of the multisection algorithm was timed. Execution times were fitted to the linear model given by (1) by a least-squares fit. The resulting measurements for the ratio s/r are listed in Table 2, together with the asymptotic rate in MFLOPS, and the resulting optimal number of points according to (2). The implementation was in FORTRAN, and all compilers vectorized the two if-statements

TABLE 2
Measurements of s/r for several vector processors.

Machine	Compiler	MFLOPS	$\frac{s}{r}$	p_{opt}
CRAY-2	cft2	29.5	9.8	7
CRAY-2	cft77	28.1	17.4	10
Alliant FX/8 (1CE)	fortran -Ogv	1.0	7.6	6
Alliant FX/8 (4CE)	fortran -O	3.4	60.0	25
Convex C1-XP	fc -O2	1.5	10.3	7

TABLE 3
Comparison bisection versus multisection.

Machine	Bisection	Multisection	p_{opt}	Ratio
CRAY-2 (cft2)	0.299	0.098	13	3.05
CRAY-2 (cft77)	0.293	0.092	18	3.18
Alliant FX/8 (1CE)	4.102	1.894	6	2.17
Alliant FX/8 (4CE)	7.164	1.881	37	3.81
Convex C1-XP	3.532	1.285	13	2.75

within the loop. The multisection loop was timed for lengths up to 2,048 iterations (i.e., multisection points). All times are averages over 50 runs, and all results reported are for 64 bit arithmetic. All machines are at the NASA Ames Research Center in Moffett Field, California.

Table 2 also lists the results for four processors of an Alliant FX-8. These are included because the model above carries over easily to this situation and yields an example of a machine with a high ratio s/r. The CRAY-2 in the benchmark is a newer model with the faster dynamic random access memory (DRAM). Results on an older CRAY-2 with the slower DRAM were not significantly different to warrant their inclusion here.

The results in Table 2 are only timing information for the simple inner loop of the multisection algorithm. They do not indicate the potential speedup using a multisection algorithm on these machines. This information is provided in Table 3. The execution times in Table 3 were obtained from running the multisection algorithm on a random tridiagonal matrix of order 3,000. The multisection algorithm was used to compute the smallest eigenvalue of the matrix. Table 3 lists both the execution time (in seconds) for bisection ($p = 1$) and for multisection with the optimal p.

The results in Table 3 show the multisection algorithm with optimal number of bisection points to be about three times faster than the bisection algorithm. The optimal p_{opt} in Table 3 has been obtained by inspection from runs of the multisection algorithm, with p varying from 1 to 60. The p_{opt} from Table 3 is different from the predicted value in Table 2. The execution times in Table 3 take the whole subroutine into account, whereas Table 2 is based on the evaluation of the simple multisection loop alone. The theoretical analysis gives an approximate prediction of the number of optimal multisection points, with the numbers in Table 2 giving an underestimate.

The results in Table 3 demonstrate that multisection is considerably more efficient than bisection for vector processors and therefore should be considered in the design of LAPACK. Some other closely related questions, for example, the use of asymptotically faster methods such as zeroin and when to switch from multisection to QR, are beyond the scope of this timely communication, but also require reevaluation in the light of our new results.

REFERENCES

[1] H. Bernstein and M. Goldstein, *Optimizing Givens' algorithm for multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 601 – 602.

[2] ——, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Tech. Rep., Courant Institute, New York University, 1985.

[3] J. J. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenvalue problem*, Numer. Math., 36 (1981), pp. 177 – 195.

[4] J. Demmel, J. Du Croz, S. Hammarling, and D. Sorensen, *Guidelines for the design of symmetric eigenroutines, SVD, and iterative refinement and condition estimation for linear systems*, Tech. Report MCSD 111, Argonne National Laboratory, February 1988.

[5] J. J. Dongarra and D. C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139 – s154.

[6] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines - EISPACK Guide Extension*, Lecture Notes in Computer Sciences, Vol. 51, Springer-Verlag, Berlin, New York, 1977.

[7] I. Ipsen and E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Tech. Report YALEU/DCS/RR-548, Yale Univ., Dept. of Computer Science, July 1987.

[8] S. Lo, B. Philippe, and A. Sameh, *A multiprocessor algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist.. Comput., 8 (1987), pp. s155 – s165.

[9] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980.

[10] D. Scott and J. Rattner, *The scalability of Intel concurrent super computers*, in Proc. Third International Conference on Supercomputing, International Supercomputing Institute, Boston, 1988, pp. 121 – 125.

[11] Y. Shih and A. Kernek, *A new generation in parallel processing systems*, SuperComputing, Winter (1988), pp. 5 – 28.

[12] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Sciences, Vol. 6, Springer-Verlag, Berlin, New York, 1976.